Temporal planner=nonlinear planner+time map manager

Eric Rutten, Joachim Hertzberg

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

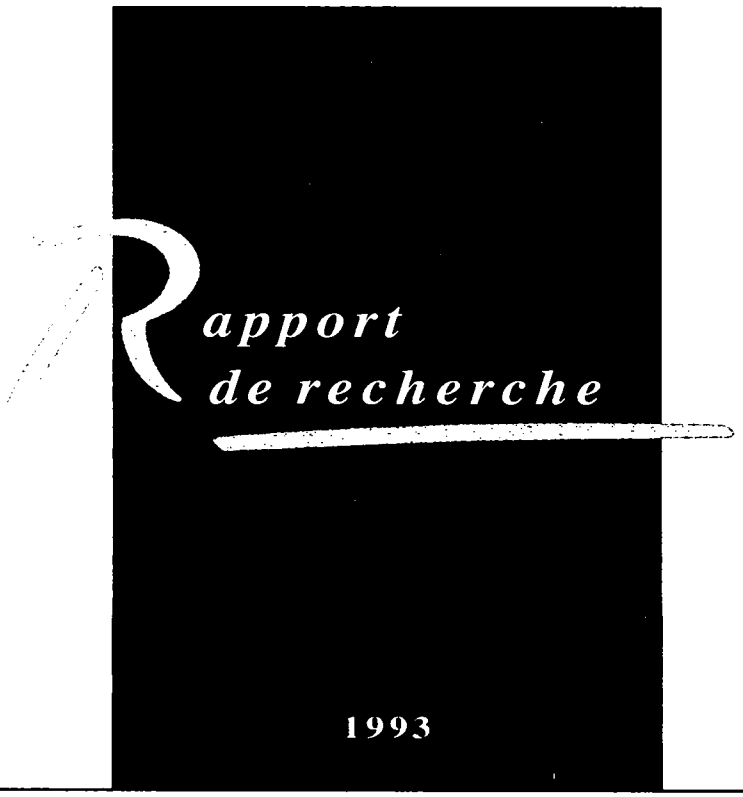
Temporal planner = nonlinear planner + time map manager

Eric RUTTEN
Joachim HERTZBERG

N° 1843
Janvier 1993

PROGRAMME 2

Calcul Symbolique,
Programmation
et Génie lociciel



IRISA

INSTITUT DE RECHERCHE
EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES
Campus Universitaire de Beaulieu
35042 RENNES CEDEX FRANCE
Tél: 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Publication Interne n°698
Programme 2
18 pages

Rennes, le 22 janvier 1993

Temporal Planner = Nonlinear Planner + Time Map Manager[1]

Eric Rutten[2]
IRISA/INRIA
Campus de Beaulieu
35042 Rennes, France
e-mail: rutten@irisa.fr

Joachim Hertzberg[3]
GMD, AI Research Division
P.O. Box 1316
5205 Sankt Augustin 1, F.R.G.
e-mail: hertzberg@gmd.de

Abstract

We describe how you end up in a temporal planner by building a classical (non-temporal) nonlinear planner on top of a time map management system. In particular, we motivate some requirements for the underlying time map manager, and describe the distribution of labour between the two components.

Planificateur Temporel = Planificateur Non-Linéaire + Gestionnaire de Contraintes Temporelles

Résumé

On décrit comment on obtient un planificateur temporel en construisant un planificateur non-linéaire classique (non-temporel) sur un système de gestion de contraintes temporelles. En particulier, on explicite les prérequis concernant le gestionnaire de contraintes temporelles sous-jacent, et on décrit la distribution des traitements entre les deux composants.

---

[1] This is an expanded version of a paper that has appeared in the proceedings of the ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation, Pisa, 1992, p.159–168

[2] This work was performed during a stay at GMD, supported by an ERCIM fellowship; ERCIM, the European Research Consortium in Informatics and Mathematics, consists of the following organizations: CNR (Italy), CWI (The Netherlands), FORTH (Greece), GMD (Germany), INESC (Portugal), INRIA (France), RAL (U.K.), SINTEF (Norway)

[3] This work is partially funded by the German Federal Ministry for Research and Technology (BMFT) in the joint project TASSO under grant ITW8900A7. TASSO is also part of the GMD Assisting Computer (AC) *Leitvorhaben*.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(U.R.A. 227) UNIVERSITÉ DE RENNES I I.N.S.A. DE RENNES

INRIA
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE
ET EN AUTOMATIQUE - UNITÉ DE RECHERCHE DE RENNES

1 The Problem Addressed

As reported [Dean and McDermott, 1987], one of Dean's motivations to start research on systems for time map management in the early mid-eighties was the idea to use them for implementing planners. Since then, both planning and time map management have been the subject of continuing interest, and a number of time map management systems and quite a large number of linear or nonlinear planners of the classical line have been built. However, only very few of these planners actually use a time map manager, notable exceptions being, e.g., FORBIN [Dean *et al.*, 1988] and O-PLAN2 [Drabble and Kirby, 1991]. Given that both classical planning and time map management are relatively well understood, that should be a surprise: Why don't more people use a technique that is available and can be of obvious help?

One of the reasons for that may be that the interplay between a time map manager and a nonlinear planner using it is non-trivial, imposing constraints and requirements on the functionalities of both of them, and on the distribution of labour between them. In this paper, we describe the problem and propose a solution, thereby proposing a modular architecture of a temporal, nonlinear planner. What we describe is implemented in the experimental planner TRIPTIC [Rutten, 1991].

The paper is organized as follows. After giving a sketch of the relevant notions of time maps and plans in section 2, we first describe in section 3 how to map plans into time maps statically, and then turn in section 4 to the questions of how to generate these plans and how to distribute the planning effort between time map manager and planner. Section 5 concludes.

2 Basic Concepts for Time Maps and Plans

In this section, we very briefly recapitulate the basic notions of time maps and plans as far as they are relevant for this paper. The reader who is fit in the respective areas may skip the respective subsections, or just browse through to pick up the terminology. Examples for the concepts described here can be found in subsequent sections.

2.1 Time Maps and Time Map Management

As basic references for work on time map management, and for more exact definitions and more comprehensive explanations, take, e.g., the original work by Dean [Dean, 1985; Dean and McDermott, 1987], or work describing the time map management system MTMM [Materne, 1991; Groß *et al.*, 1992], which we have used for implementing the work reported here, and the theoretical basis of which is presented in the following in a simplified form.

A time map is a database containing temporally qualified *tokens.* A token represents the fact that an event of some *type* is true over some interval. (We will notate tokens with the upper capital T or other capitals lexically close to it.) A token T is represented in the time map by representing

- its token type,
- its start point σ_T and its end point η_T,
- the duration of T, expressed as the *temporal relation* between σ_T and η_T

(We denote points with lowercase greek letters throughout the text, where σ and η with the respective subscripts denote token start and end points, respectively.)

A temporal relation $\varphi[x, y]\psi$ between two points φ, ψ is a pair $[x, y], x \leq y$, of rational numbers, where x (resp. y) is to be interpreted as the lower (resp. upper) bound of the temporal distance between φ and ψ. (We sometimes call temporal relations *constraints* here.) If $x = y$, then the relation is *exact,* otherwise it is called *vague.* Note that temporal relations are defined between arbitrary points, not just start and end of the same token. Absolute time information may be expressed as the temporal relation to some reference point, say, *midnight.* In general, there may be many different temporal relations $[x_i, y_i]$ between a pair of points; hence, the *minimized relation* $\varphi\langle x, y\rangle\psi$ between φ and ψ is defined by $x = \max(x_i)$ and $y = \min(y_i)$. The minimized relation between the *start* and *end* of a token is the *token interval.* A time map is *constraint inconsistent*, if $\varphi\langle x, y\rangle\psi$ and $x > y$ holds for any two points φ, ψ.

The time map *management* system makes no deductions w.r.t. its tokens, i.e., from a token P of type $\mathcal{P}$, another token Q of type $\mathcal{Q}$ and some implication $\mathcal{P} \wedge \mathcal{Q} \rightarrow \mathcal{R}$, it will not be inferred that another token of type $\mathcal{R}$ is valid over the overlapping interval of P and Q. If such deductions are required, they must be done outside the time map manager. However, as one of its central features, a time map manager handles tokens of contradictory types, where two token

types $\mathcal{S}$ and $\mathcal{T}$ must and can be explicitely defined as contradictory. This leads to a second form of inconsistency, called *token inconsistency.* Two tokens are *inconsistent,* if they are contradictory and overlapping, where two tokens S, T with $\sigma_S\langle ssmin, ssmax\rangle\sigma_T$ and $\sigma_T\langle semin, semax\rangle\eta_S$ *overlap,* iff $ssmax \geq 0$ and $semax \geq 0$, or analogously with S ant T interchanged.

Now, the question is what to do if a time map is constraint inconsistent or token inconsistent. As to constraint inconsistency, nothing can be done; it can be remedied only by asking the user (which can be a human or an artificial system, e.g., a planner) to withdraw constraints. Token inconsistency is more interesting. Depending on whether minimized relations involved are exact or vague, tokens may overlap *definitely* or *non-definitely.* Two tokens S, T as above overlap definitely iff $ssmin \geq 0$ and $semin \geq 0$, or ananlogously for S and T interchanged; they overlap non-definitely, iff they overlap and do not overlap definitely. If contradictory tokens definitely overlap, then nothing can be done to resolve the contradiction in the time map, and it is again up to the user to withdraw constraints.

If they non-definitely overlap, *persistence clipping* is possible: a *clipping constraint* constrains the end point of the token that starts earlier, to be before the start point of the token starting later, or analogously if only the ends of tokens are comparable. This pre-order may be undetermined. In this case, the user must choose how to clip. Moreover, the user may choose to postpone clipping even if it is unique; hence, the time map manager does not guarantee token consistency in every system state. (This is discussed in detail in [Groß *et al.*, 1992].) We will see that this postponing is an important feature of a time map manager if you use it for planning.

2.2 Plans

We now turn to recapitulating some concepts of classical plans. Note that the issue here is not that much planning, i.e., the process of generating plans, but we focus on the more declarative concepts related with planning, omiting nearly all details concerning, e.g., control issues, abstraction, or uncertainty management, which are important for planning. For more details about planning as a whole, consult an survey paper like [Hendler *et al.*, 1990], a textbook like [Hertzberg, 1989], or a source book like [Allen *et al.*, 1990]. Moreover, even the declarative concepts are simplified, compared to more sophisticated planners like SIPE [Wilkins, 1988]. In particular, we assume that all conditions in

plans be *propositional* literals.

A *plan* is a pair $(\Theta, \prec)$, where $\prec$ is a strict (possibly partial) ordering on Θ and Θ is a *task* set, where a task is an instance of an *operator,* which is the representation of some class of actions in the real world. Operators are defined by their *preconditions* and *postconditions,* where the preconditions must hold for an instance of the operator to be applicable, and the postconditions will be true after its execution. In the case of *temporal* planning, i.e., planning respecting numerical time information, an operator is given a *duration.* A plan contains two dummy tasks *start* and *goal*, where *start* is the $\prec$-smallest, and *goal* the $\prec$-largest element in the plan. The postconditions of *start* are a description of the initial situation, i.e., the situation in which the plan execution starts; the precondition of *goal* is a (partial) description of the set of features that shall be true in a desired situation.

If a task s generates a condition C and s is not ordered after task t, i.e., $t \not\prec s$, then C *possibly persists* until t, if in some linearization of the tasks in the whole plan, C is not destroyed between s and t, i.e., there is a linearization $\prec'$ such that $s \prec' t$, and there is no task u generating $\neg c$ such that $s \prec' u \prec' t$. There is a *dependency* between s and t w.r.t. C, if C possibly persists from s until t and C is among the preconditions of t. There is a *conflict* between a task t_d (the destroyer task) and a dependency between t_p (the producer task) and t_n (the needer task) wrt. C, if there is a linearization $\prec'$ of $\prec$ such that $t_p \prec' t_d \prec' t_n$, i.e., the destroyer can be linearized between the producer and the needer. See [Hertzberg and Horz, 1989] for a more extensive treatment of the notions of dependency and conflict; dependencies are sometimes called *causal links,* e.g., [McAllester and Rosenblitt, 1991].

So, there are in fact two orderings in a plan, the difference between which is important for this paper. First, there is the temporal ordering $\prec$, which means that a task s must be executed temporally before t if $s \prec t$. Second, there is the ordering induced by dependencies. Note that temporal and dependency ordering are different as independent tasks may be ordered in time. However, if t depends on s, then s must not follow t in time; hence, the temporal order is stronger than the dependency order.

Pragmatically, there is another notion that is relevant here when generating plans, which is classically called *helpful interaction.* We said that this is just a pragmatic notion because a helpful interaction is a plain dependency, with the only feature that it has emerged occasionally in the process of generating a plan: a precondition C of some task t turns out to be produced by some other

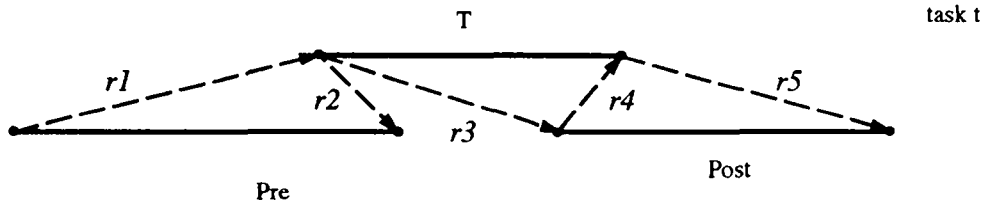


Figure 1: The temporal constraints for a task. Bold lines represent token intervals, arrows represent constraints. Numerical information about temporal relations is not shown.

task s (and possibly conflicted by some third task) without having explicitely cared about it. We just introduce this notion because it will be of interest in the planner TRIPTIC below.

3 Mapping Plans into Time Maps

We will now see how these two kinds of concepts can be integrated into a temporal representation of plans, first giving the representation of tasks, and then seeing how several tasks can interact, in particular in conflicts.

3.1 Representing tasks in a time map

A task t is represented in its temporal dimension by a token T as illustrated in fig. 1. Its duration is represented by the duration of T. (In the following, the type of task tokens is unimportant; simply assume that all task tokens are of some arbitrary type, say, $\mathcal{T}$. Moreover, whenever this causes no confusion, we will not distinguish between a task and a task token representing it.) Other constraints between points are given in terms of "before" or "after", corresponding respectively to the quantitative temporal relations $[0, +\infty]$ and $[-\infty, 0]$.

A condition C is represented as a token of some type $\mathcal{C}$. Preconditions of T are constrained to be true at the start of T: i.e., for each precondition Pre, its start σ_{Pre} is before σ_T (relation r_1 in fig. 1) and η_{Pre} is not before σ_T (relation r_2). Note that nothing is required as to the overlapping of Pre and T; preconditions both overlapping tasks (partially or completely) and preconditions terminated by the start of the task can be represented. Postconditions are

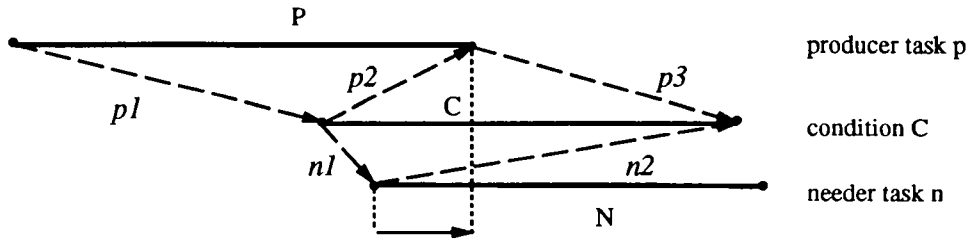


Figure 2: A flexible temporal constraint for postconditions with the interval of parallel execution of producer and needer task shown between dashed lines.

constrained to be true at the end of the task, i.e., for a postcondition *Post*, σ_{Post} is after σ_T (relation r_3) and not after η_T (relation r_4): this represents the fact that the effects are produced during the execution of the task, and η_{Post} is after η_T (relation r_5): this represents that the effects still hold at the end of the action. Usually, pre- and postconditions will overlap; if the postcondition is of a type contradictory to that of a condition in the environment, e.g., when it is the negation of a precondition, then the "deletion" effect will be handled by the persistence clipping mechanism of the TMM. A plan is represented by the set of tasks, and the relations between the start and end points of their tokens.

Note that this action representation is more expressive than, e.g., DEVISER's [Vere, 1983] in that it deals symmetrically with task start and end times and allows to state temporal relations between arbitrary points. E.g., it is possible to encode numerical information about the time of occurrence of each individual postcondition directly.

These constraints are given explicitely between the task token and tokens representing conditions in the environment. This entails, however, indirect constraints between token intervals of different tasks, especially through the conditions of dependencies, that are preconditions of a *needer* task, and postconditions of a *producer* task: the corresponding constraints are illustrated in fig. 2, showing that the *producer* need not be terminated for the *needer* to start, which is to be interpreted as a partially overlapping execution.

Note that this constraints scheme involves choices that may be discussed. In particular, constraining effects to begin during the task's execution intervals does not allow to account for delayed effects (starting after the end of the task). However, these considerations are adaptable, and depend on the application

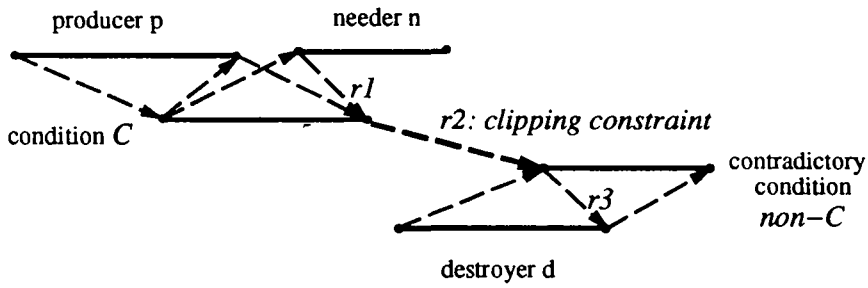


Figure 3: The temporal constraints involved in a conflict, including the clipping constraint for solving it.

domain.

Note further that one could use fully quantitative constraints to dispose the intervals relatively, with greater precision, defining for each pre- or post-condition how long after or before the start or end of the task token it starts or ends. As already said, the qualitative relation "before" is quantitatively represented by the weakest numerical constraint, $[0, +\infty]$, which is compatible with further, more precise, numerical information. This way, precise quantitative information is just a special case of the quantitative relations we will use henceforth.

3.2 Conflicts

Conflicts are treated by the time map manager, i.e., by temporal constraints, especially by persistence clipping between contradicting tokens intervals, as illustrated in fig. 3. The figure should also make plausible that the classical handling of conflicts as presented in section 2.2 is a special case of the view presented here, where tasks have unit durations.[1] We now briefly explain how conflicts are handled.

Detection and solving. Given our time map representation of actions, preconditions, and postconditions, conflicts show in the time map as overlapping

[1]Note that there is a problem in avoiding an overlap of contradictory tokens in the "clipping point". We omit this problem like we have omitted the possibility of defining the borders of token intervals to be open or closed. For example, it must be avoided to specify a task of zero duration with a contradictory pair of pre- and postconditions. [Horz, 1993] describes some intricacies of the problem and their ramifications.

contradictory condition tokens, and they can be solved by clipping the persistence of one of the condition tokens involved: If there is a relation possibly ordering some of their extremities, then the interval that is possibly before the other is linearized before the other. Note that if *any* of the points of an interval is before (after, resp.) a point of the other interval, then there is only one allowable clipping. In effect, *all* the points of the clipped interval will be before (after, resp.) all the points of the other interval. In some cases, the condition token and contradictory condition token involved in a conflict are unordered; as an example consider the situation in fig. 3 without the clipping constraint $r2$. In this case, there are two linearizations of the two tokens, both of which solve the conflict. It may depend on third tasks in the plan which linearization is preferable; if there is, e.g., a needer of the contradictory condition *non-C* after the needer N in fig. 3, then *non-C* should be linearized *after C* as by $r2$. To avoid early overcommitments in ordering conditions, a decision about which linearization to choose should be postponed in such a case, temporarily leaving the conflict in the plan.

Note, however, that not every pair of overlapping contradictory tokens represents a conflict. Consider, e.g., the case that two non-ordered tasks s and t produce P and *non-P*, respectively, both of which are not used by any of the tasks later in the plan. This would yield overlapping contradictory tokens; however, persistence clipping would mean to order the respective producers, which would result in an unnecessary ordering constraint in the tasks of the plan: given that there is no needer of the conditions P and *non-P*, it is unimportant which of the two holds after executing both s and t. So, in order to avoid overconstraining the task ordering, the time map manager must tolerate to postpone clipping although being aware of overlapping contradictory tokens—be they part of a conflict or not.

Influence on the relation between tasks. Clipping contradictory tokens entails a particular temporal relation between the needer and the destroyer of a condition. This relation is obtained indirectly through the condition in the environment around which the conflict occurs, as illustrated in fig. 3: it involves the precondition-typical relation between the start of the needer and the end of the condition ($r1$), the clipping constraint $r2$ between the condition and its contradictory condition, and the postcondition-relation between the contradictory condition and its producer, i.e., the destroyer of the condition ($r3$). One way of describing the meaning of this relation is to say that *"the needer must begin before the destroyer ends"*, i.e., that the needer must have

finished needing the condition, before the destroyer can destroy it by producing a contradictory effect.

This treatment of conflicts has the advantage of centering the constraining of relations between tasks on the involved conditions in the environment. As a consequence, two other tasks involved in a conflict regarding the same tokens will take profit from the solving through the persistence clipping. The conflict has been solved once for all, and "encoded" in the constraints between the conditions tokens.

4 A Time Map Based Temporal Nonlinear Planner

We now sketch the temporal planner TRIPTIC which is presented more completely elsewhere [Rutten, 1991]. We will concentrate on its temporal aspects, and present only informally a sketch of the generation algorithm. Using the ideas presented in the previous section, it is built on top of the time map manager MTMM [Materne, 1991; Materne and Hertzberg, 1991], which in turn implements the concepts presented in section 2.1. TRIPTIC generates non-linear plans, interpreted as a set of tasks executable in parallel, i.e., on overlapping intervals, when the constraints allow so.

The distribution of labour between the two components is as follows:

- The planner handles the logical dependencies between tasks and their organization towards the goal of the planning problem, respecting interactions. The important point is that the plan structure does not contain any explicit information about the temporal precedences of tasks in general: it just contains their logical dependencies.

- The time map manager handles all the temporal information, in particular all the quantitative constraints between points. As far as this information concerns relations between tasks, it is given to the time map manager by the planner. Other temporal relations are determined by the time map manager and the model of the environment. In particular, the time map manager is responsible for detecting possible helpful interactions, and for detecting and handling conflicts. This temporal part implements the concepts of the previous section.

We will now explain how the planner works, interacting with the temporal part by posting new constraints or querying the existing ones.

Task and plan representation. Seen from the planner's point of view, a task is a four-tuple: $\langle t, d_t, Pre, Post \rangle$, where t is the name of the task, d_t is its duration, the preconditions set Pre is a set of conditions, as is the set of postconditions $Post$.

The basic element of a plan is a dependency $\langle t, C, n \rangle$, defined by a producer task t (which may be the virtual task *start*), the condition produced C, and a needer task n (which might be the virtual task *goal*). An unsolved dependency has no known producer: we will note this by a $\perp$ for the producer: $\langle \perp, C, n \rangle$. An unneeded produced dependency has the same structure, but, symmetrically, with no specified needer:$\langle t, C, \perp \rangle$.

The dependencies contained in the plan are classified into four groups. The *unsolved dependencies* are those that are needed in the plan and for which there is no producer task in the plan yet. The *possibly solved dependencies* are those for which there is already a producer in the plan such that it is possible that the dependency is solved by a helpful interaction, i.e., there is a producer task and there is no constraint forbidding it to be the producer for that particular dependency.

The *solved dependencies* are those needed by a task in the plan, and for which a producer is present and identified in the plan. The *produced dependencies* are all the postconditions of all the tasks in the plan, including the effects that are not needed by another task in the plan. This allows to know about all the effects of the tasks in the plan, whereas the solved dependencies indicate only the dependencies directly relevant to the plan, i.e. needed for its coherence. In this sense, the solved dependencies are a subset of the produced dependencies.

Plan generation. An informal sketch of the algorithm is given in table 1. The plan is initialized with unsolved dependencies for all conditions of the goal situations (with needer task *goal*), and produced dependencies for all conditions of the initial situation (with producer task *start*). We assume that the time map contains tokens *Start* and *End* that represent the *start* and *end* tasks in the plan, respectively.

The plan is generated backwards, expanded in a tree-like fashion, starting from the goal dependencies, as in classical non-linear planning. There are three points where the planning algorithm and the time map manager interact, which we will briefly describe: the input of constraints, the handling of conflicts, and

while there are unsolved dependencies
do choose one of them: $u = \langle \perp, C, n \rangle$;
choose a task producing C (i.e., $\langle t, d_t, Pre, Post \rangle$ such that $C \in Post$) ;
input the new temporal constraints for t and u ;
look for conflicts and handle them ;
replace u in the plan by the solved dependency $\langle t, C, n \rangle$;
add unsolved dependencies in the plan for all $D \in Pre$: $\langle \perp, D, t \rangle$;
criticize the plan
end

Table 1: A sketch of the generation algorithm, with calls to the TMM (in boldface).

the critique of the plan.

The input of the new temporal constraints in the time map manager, for a task $\langle t, d_t, Pre, Post \rangle$ and a dependency $u = \langle \perp, C, n \rangle$, is described in table 2, and realizes the relations shown in fig. 2.

Conflicts are handled as described in section 3.2. They are treated by the time map manager, thus the planner is releaved from managing these interactions.

The critique of the plans concerns mainly helpful interactions, as defined in section 2.2. It involves checking the unsolved dependencies, in order to see if there exists a producer for their condition, that could possibly become the producer solving the dependency. The criterium is that such a producer task p and the condition it produces C should be able to satisfy the constraints with the needer task n, as illustrated in fig. 2 in the time map representation. Thus, a helpful interaction is possible between a token P (representing task p) and task N (representing task n) w.r.t. a condition C, iff $\neg(\sigma_N$ *before* $\sigma_C \vee \sigma_N$ *after* $\eta_C)$. This is tested, for each unsolved dependency and each producer of the concerned condition, by querying the time map manager.

If this helpful interaction is possible, it can be realized, by inputing the corresponding constraints between C and N in the time map, namely the pre-condition relations n_1 and n_2 in table 2 (see also fig. 2). In TRIPTIC this is done in a delayed way, in order to control the generation algorithm. Plan expansions are made only for unsolved dependencies for which there is *no* possible helpful interaction. The others are considered to be *possibly solved.* But in order to avoid premature commitments and over-constraining, the realization of the interactions (i.e. input of the temporal constraints) is postponed until

input token T of duration d_T for task t ;
constrain σ_T *after* η_{Start} ;
**** constrain C as a precondition of n as in fig. 2, i.e.:*
constrain σ_C *before* σ_n ; *(relation n_1)*
constrain η_C *after* σ_n ; *(relation n_2)*
**** constrain all postconditions of t, i.e.:*
for each P' **in** *Post* **do**
 input token P for condition P' ;
 constrain σ_P *after* σ_t ; *(relation p_1)*
 constrain σ_P *before* η_t ; *(relation p_2)*
 constrain η_P *after* η_t ; *(relation p_3)*
end ;

Table 2: The constraints for a task $\langle t, d_t, Pre, Post \rangle$ solving dependency $u = \langle \perp, C, n \rangle$ in the plan.

there is no more unsolved dependency left.

The plan expansion ends when there are no more unsolved dependencies. Then, possibly solved dependencies are taken into account, and the helpful interactions are "realized": the corresponding constraints are added to the plan. As these dependencies have been selected as being possibly solved, there was no constraint in the plan causing conflicts involving them. However, introducing additional constraints in the plan can cause helpful interactions for the other possibly solved dependencies that are not yet realized. Therefore, the plan is also criticized between realizations.

Note that the detection and solving of conflicts as described above, is performed without intervention on the plan structure, and especially without breaking already solved dependencies.

5 Conclusion

We proposed a way to build a temporal planner as a nonlinear planner on top of a time map manager (TMM). We described the distribution of the work between the two components, showing that all the temporal information can be treated by the TMM, which leaves the planner with its essential work: deal with dependencies. Thus, the planner is simplified in that some of the work it classically does, is here transmitted to the temporal level, i.e., to the TMM.

The standard TMM mechanism of persistence clipping proves very useful for detecting and handling conflicts; the TMM used must, however, tolerate to work on time maps that are not token consistent, which forbids to use Dean's original TMM [Dean and McDermott, 1987]. Note that this is a weakness in Dean's original work, rather than a weakness of our particular approach to mapping plans into time maps. As discussed in section 3.2, not every pair of overlapping contradictory tokens in a plan should be resolved; in the TMM, however, you are forced to do so.

The non-linear planner gives only a partial order between tasks: combined with the temporal representation, this allows to interpret a plan as a set of possibly overlapping parallel tasks. Using a TMM has the advantage that it is possible to use quantitative information (numerical delays, durations, etc.); the qualitative relation "before" represented as $[0, +\infty]$ is compatible with this more precise data.

Possible extensions of the planner TRIPTIC include using the TMS functionalities in the TMM: the possibility of taking back constraints in the TMM can be useful when backtracking in the search process of the planner. The way numerical quantitative information could be input simply and shortly (i.e., without having to say everything) in the task descriptions should be improved. Also, the resulting time map, after a plan generation, consists of a huge temporal data-base: user-friendly ways of outputting it, and exploiting the information there, would be precious. The search strategy of TRIPTIC is presently experimentally naïve and would deserve more attention.

References

[Allen *et al.*, 1990] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* Morgan Kaufmann, San Mateo, CA, 1990.

[Dean and McDermott, 1987] T.L. Dean and D.V. McDermott. Temporal data base management. *J. Artificial Intelligence*, 32:1–55, 1987.

[Dean *et al.*, 1988] T.L. Dean, R.J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *J. Computational Intelligence*, 4:381–398, 1988.

[Dean, 1985] T.L. Dean. Temporal imagery: An approach to reasoning about time for planning and problem solving. Tech. Report 433, Yale University, Comp. Sci. Dept., 1985.

[Drabble and Kirby, 1991] B. Drabble and R. Kirby. Associating AI planner entities with an underlying time point network. In J. Hertzberg, editor, *European Workshop on Planning. EWSP '91, Sankt Augustin, FRG, March 1991, Proceedings*, pages 27–38. Springer Verlag (Lecture Notes in AI vol. 522), 1991.

[Groß *et al.*, 1992] E. Groß, J. Hertzberg, S. Materne, and H. Voß. On clipping persistence (or whatever must be clipped) in time maps. Technical Report KI-NRW-92-20, Forschungsverbund "KI in NRW", 1992. Submitted for publication.

[Hendler *et al.*, 1990] J. Hendler, A. Tate, and M. Drummond. AI planning: Systems and techniques. *AI Magazine*, 11(2):61–77, 1990.

[Hertzberg and Horz, 1989] J. Hertzberg and A. Horz. Towards a theory of conflict detection and resolution in nonlinear plans. In *Proc. IJCAI-89*, pages 937–942, 1989.

[Hertzberg, 1989] J. Hertzberg. *Planen. Einführung in die Planerstellungsmethoden der Künstlichen Intelligenz.* BI Wissenschaftsverlag, Mannheim u.a., 1989.

[Horz, 1993] A. Horz. Relating classical and temporal planning. In A. Horz, editor, *Beiträge zum 7. Workshop "Planen und Konfigurieren"*. GMD Arbeitspapier, 1993. In press.

[Materne and Hertzberg, 1991] S. Materne and J. Hertzberg. MTMM – correcting and extending time map management. In J. Hertzberg, editor, *European Workshop on Planning. EWSP '91, Sankt Augustin, FRG, March 1991, Proceedings.* Springer Verlag (Lecture Notes in AI vol. 522), 1991.

[Materne, 1991] S. Materne. *MTMM – Ein System zur Verwaltung von Zeitverhältnissen.* GMD-Bericht Nr. 193. R. Oldenbourg Verlag, München, 1991.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. AAAI-91*, pages 634–639, 1991.

[Rutten, 1991] E. Rutten. A temporal non-linear planner: TRIPTIC. Arbeitspapier 582, GMD, 1991.

[Vere, 1983] S.A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-5:246–267, 1983.

[Wilkins, 1988] D. Wilkins. *Practical Planning. Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.

Liste des publications internes Irisa 1993

PI 694 MECANISMES D'ABSTRACTION DANS UNE REPRESENTATION DE LA CONNAISSANCE CENTREE OBJET (R.C.O.)
Stéphane LE PEUTREC, Sophie ROBIN
Janvier 1993, 40 pages.

PI 695 DETECTION DE SEQUENCES ATOMIQUES DE PREDICATS LOCAUX DANS LES EXECUTIONS REPARTIES
Michel HURFIN, Noël PLOUZEAU, Michel RAYNAL
Janvier 1993, 16 pages

PI 696 SEMI-UNIFIED CACHES
Nathalie DRACH, André SEZNEC
Janvier 1993, 18 pages.

PI 697 CONCEPTS ET PROBLEMES DE L'ALGORITHMIQUE REPARTIE
Michel RAYNAL
Janvier 1993, 18 pages.

PI 698 TEMPORAL PLANNER = NONLINEAR PLANNER + TIME MAP MANAGER
Eric RUTTEN, Joachim HERTZBERG
Janvier 1993, 18 pages.

Unité de Recherche INRIA Rennes
IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399

RR-1843